

On How Turing and Singleton Arc Consistency Broke the Enigma Code

Valentin Antuori, Tom Portoleau, Louis Rivière, Emmanuel Hébrard

On How Turing and Singleton Arc Consistency Broke the Enigma Code

Valentin Antuori ✉
Renault, LAAS-CNRS, Université de Toulouse, CNRS, France

Tom Portoleau ✉
LAAS-CNRS, Université de Toulouse, CNRS, France

Louis Rivière ✉
LAAS-CNRS, Université de Toulouse, CNRS, ANITI, France

Emmanuel Hebrard ✉
LAAS-CNRS, Université de Toulouse, CNRS, ANITI, France

Abstract

In this paper, we highlight an intriguing connection between the cryptographic attacks on Enigma's code and local consistency reasoning in constraint programming.

The coding challenge proposed to the students during the 2020 ACP summer school, to be solved by constraint programming, was to decipher a message encoded using the well known Enigma machine, with as only clue a tiny portion of the original message. A number of students quickly crafted a model, thus nicely showcasing CP technology – as well as their own brightness. The detail that is slightly less favorable to CP technology is that solving this model on modern hardware is challenging, whereas the "Bombe", an antique computing device, could solve it eighty years ago.

We argue that from a constraint programming point of vue, the key aspects of the techniques designed by Polish and British cryptanalysts can be seen as, respectively, path consistency and singleton arc consistency on some constraint satisfaction problems.

2012 ACM Subject Classification Mathematics of computing → Combinatoric problems; Mathematics of computing → Combinatorial optimization

Keywords and phrases Constraint Programming, Cryptography

Digital Object Identifier 10.4230/LIPIcs.CP.2021.13

Supplementary Material *Software (Source Code)*: `https://gitlab.laas.fr/vantuori/sacnigma`
archived at `swh:1:dir:98f5264c0b6821dbf5caf769a2ebf70e4cda5b92`

1 Introduction

Enigma was a cipher machine that had been commercialised since 1923. Breaking its code was a decisive breakthrough with a significant impact on the outcome of World War II.

The machine resembles a portable typewritter. Once configured in a particular setting agreed upon by the sender and the receiver, it can be used to encrypt a message. When typing with the machine, each letter is ciphered to a seemingly random letter indicated by a light bulb. The encrypted message, or *ciphertext*, can be deciphered by the receiver using his own Enigma machine. The code is indeed symmetric and typing the ciphertext with the same machine setting yields the original message.

The Enigma code was first broken by the mathematicians Marian Rejewski, Jerzy Różycki and Henryck Zygalski for the Polish Cipher Bureau before the war, although this method relied on a weakness due to an operating practice that was abandoned during the war. This knowledge on the machine, however, was shared with the allies and helped British cryptanalysts to break the code. To this end, Alan Turing and Gordon Welchman designed "The Bombe", an electro-mechanical device that made it possible to decipher Enigma's encrypted messages until the end of the war.

In this paper, we look back to this story from the viewpoint of constraint programming. We first describe a constraint model to break Enigma's code in Section 4. Given a portion of the original message (the *crib*), a solution of this model represents the internal settings of Enigma (the *cryptographic key*) that produced, and can decrypt, the intercepted communication. Modelling this constraint satisfaction problem was the topic of a "hackathon" held during the ACP summer school in 2020, and several participants managed to break Enigma's code during this event.[1] This problem would be deceptively tricky to tackle without a rich modelling framework, and it nicely illustrates the effectiveness of constraint programming in that respect. However, solving this model with state-of-the-art solvers appears to be challenging, which highlights the prowess that was solving this problem eighty years ago. Interestingly, it appears that both methods used by the Polish Cipher Bureau or at Bletchley Park can be equated to known concepts of consistency: *Singleton Arc Consistency* on the constraint model introduced in this paper for the latter (see Section 5), and *Path Consistency* on some precomputed constraints for the former (see Section 6).

2 The Constraint Satisfaction Problem and Consistency

A Constraint Satisfaction Problem (CSP) is a triple $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ where $\mathcal{X} = \{x_1, \ldots, x_n\}$, is a set of *variables*; $\mathcal{D} = \{D(x_1), \ldots, D(x_n)\}$ a set of *domains*; and $\mathcal{C} = \{c_1, \ldots, c_t\}$ a set of *constraints*. An *assignment* maps[2] values to variables, we write $x \leftarrow v$ for the assignment of value v to variable x. A constraint c_j is given by a pair $(S(c_j), R(c_j))$ where $S(c_j)$ is a subset of $\mathcal{X}$, and $R(c_j)$ is $|S(c_j)|$-ary relation, that is, a set of satisfying assignments of $S(c_j)$.

The assignment $A = \{x_1 \leftarrow v_1, \ldots, x_k \leftarrow v_k\}$ of the set of variables $\{x_1, \ldots, x_k\}$ is *consistent* for a constraint c if and only if its projection $\{x_i \leftarrow v_i \mid 1 \leq i \leq k \;\&\; x_i \in S(c)\}$ to $S(c)$ can be extended to an assignment in $R(c_j)$; Assignment A is *globally* consistent if and only if it is consistent for every constraint in $\mathcal{C}$; it is *valid* if and only if, for every variable x_i, we have $v_i \in D(x_i)$. A *solution* of a CSP $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ is a valid, globally consistent assignment of $\mathcal{X}$. We write $\mathcal{D}|_{x \leftarrow v}$ for the set of domains where x is mapped to $\{v\}$ and equal to $\mathcal{D}$ on all other variables, and $\mathcal{D}' \subseteq \mathcal{D}$ when $\forall x \in \mathcal{X},\; D'(x) \subseteq D(x)$.

The notion of consistency is key to solving constraint satisfaction problems. We define here three consistencies that we shall relate to historical methods for attacking Enigma's code. These definitions are standard generalisations to non-binary constraints. In particular, the definition of consistency of an assignment for a constraint as its *projection* to the constraint's scope being *extendable* to the constraint's relation is useful to generalize path consistency.

▶ **Definition 1.** *A* support *for a constraint c is a valid and consistent assignment of $S(c)$.*

▶ **Definition 2** (Arc Consistency (AC) [6]). *A variable x is* arc consistent *(AC) with respect to a constraint c if and only if, for each $v \in D(x)$, there exists a support for c that contains $x \leftarrow v$. A constraint c is AC if and only if every variable $x \in S(c)$ is AC with respect to c. A CSP $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ is AC if and only if every constraint $c \in \mathcal{C}$ is AC.*

▶ **Definition 3** (Singleton Arc Consistency (SAC) [3]). *An instantiation $x \leftarrow v$ is* singleton arc consistent *(SAC) if and only if there exists $\mathcal{D}' \subseteq \mathcal{D}|_{x \leftarrow v}$ such that the CSP $(\mathcal{X}, \mathcal{D}', \mathcal{C})$ is AC. A variable x is SAC if and only if, and for each $v \in D(x)$, $x \leftarrow v$ is SAC. A CSP $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ is SAC if and only if every variable $x \in \mathcal{X}$ is SAC.*

---

[1] `https://acp-iaro-school.sciencesconf.org/`
[2] We use functions instead of tuples to make variable ordering irrelevant.

▶ **Definition 4** (Path Support). *Given a CSP* $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ *and three distinct variables* x_1, x_2 *and* x_3*, the assignment* $\{x_3 \leftarrow v_3\}$ *is a* path support *of assignment* $\{x_1 \leftarrow v_1, x_2 \leftarrow v_2\}$ *for variable* x_3 *if and only if the assignment* $\{x_1 \leftarrow v_1, x_2 \leftarrow v_2, x_3 \leftarrow v_3\}$ *is valid and globally consistent.*

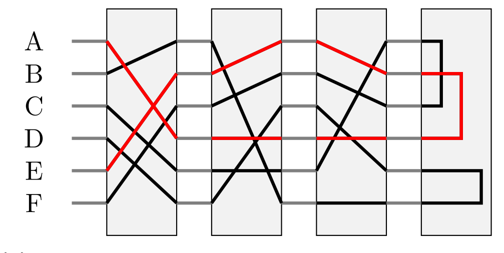
▶ **Definition 5** (Path Consistency (*PC*) [7]). *An assignment* $\{x_1 \leftarrow v_1, x_2 \leftarrow v_2\}$ *of two variables is* path consistent *(PC) if and only if it has a path support for every variable.*

A CSP $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ *is PC if and only if every valid and globally consistent assignment of every pair of variables is path consistent.*

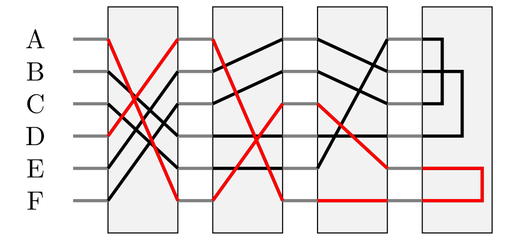
Enforcing (singleton) *AC* on a CSP $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ means finding the largest domain $\mathcal{D}'$, in the sense of inclusion as defined above, such that $\mathcal{D}' \subseteq \mathcal{D}$ and $(\mathcal{X}, \mathcal{D}', \mathcal{C})$ is (singleton) *AC*. Notice that there is a single such fix point, since the set of possible domains forms a lattice where infimum and supremum are obtained respectively by the intersection and the union.

3 The Enigma Code

In its simplest form, Enigma's encryption system is composed of three *rotor* wheels and a *reflector* wheel. A rotor wheel can be seen as a *simple substitution cipher* whereby every letter is mapped to a given letter, forming a permutation of the alphabet. The signal then goes through the two other wheels, then through the reflector and finally through the three rotors but backwards. Figure 1a illustrates, on a reduced alphabet, the rotor wheels (first three boxes) and reflectors (last box) wiring the input keyboard to an output system composed of lightbulbs indicating the substituted letter. In this case, pressing the key `A` eventually lights the bulb `E`. In other words, this mechanism is a simple substitution cipher whereby the alphabet is applied a permutation, e.g., (`AE`)(`BD`)(`CF`) in Figure 1a. Notice that the reflector wheel is symmetric and antireflexive. As a result, the overall permutation is symmetric and the same machine can therefore be used to decipher: pressing the key `E` lights bulb `A`. Moreover, it is antireflexive: no letter is mapped to itself, which proved to be a weakness.



(a) Original rotors and reflector wiring.



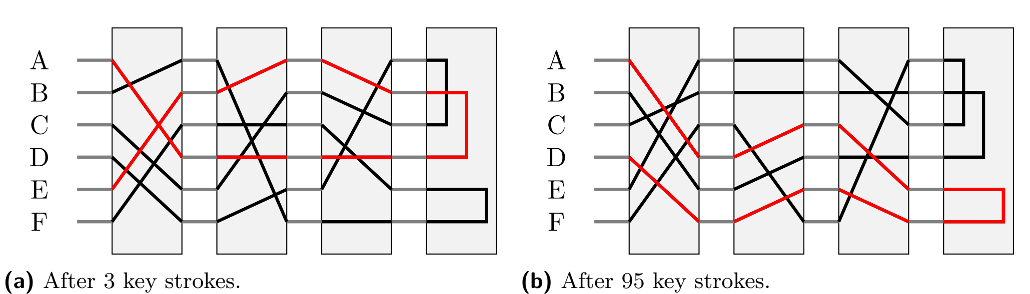
(b) The same machine after one key stroke.

Figure 1 Illustration of Enigma's rotors and reflector.

However, the feature that made Enigma such a strong cryptographic system is that rotors are not static: they advance at every key stroke. More precisely, at every key stroke the "fast" rotor (leftmost) advances one step. Figure 1b shows the same machine after one key stroke. The fast rotor is shifted down by one position: the wire that was connecting `B` to `A` nows connects `A` to `F`, and so forth. As a result, pressing the same letter `A` now lights bulb `D`.

Moreover, when the fast rotor has completed a turn and is back to its reference position, a turnover notch is activated, and the middle rotor advances one step. Similarly, when the middle rotor has completed a turn, the "slow" (rightmost) rotor advances one step. Figure 2 shows the same machine after 6 key strokes (the fast rotor has made a full turn and is back

to its original position and the middle rotor has advanced 1 step), or after 95 key strokes (the fast rotor has advanced 5 steps from its reference position, the middle rotor 3 steps and the slow rotor 2 steps). The simple substitution cipher in the latter position is (AD)(BC)(EF).



(a) After 3 key strokes. **(b)** After 95 key strokes.

Figure 2 Illustration of the same machine as in Figure 1a.

All Enigma machines shared the same set of three rotor wheels and the same reflector wheel, however, rotors could be rearranged in any order and initialised in any position. For instance, the daily settings could be given as "312, CSP". In that case, on that day, the 3rd rotor wheel would placed on the left, the 1st wheel on the middle and the 2nd wheel on the right. Moreover, before (de)ciphering any message, the left, middle and right rotor wheels would be positionned respectively so that the letters C, S and P showed up in some windows designed for this purpose. Additionally, the positions of the turnover notches of the left and middle rotor wheels, indicating the reference position at which the next rotor to the right would advance, could also be changed. The cryptographic key shared by the sender and receiver of the message was therefore the 3! *rotor orders* and the 26^3 *rotor positions*, as well as the 26^2 *reference positions*.[3] Therefore, there are $3! \times 26^3 \times 26^2 = 72188256$ possible keys.[4]

The first versions of the machine, commercialised in the 1920's operated on those principles, and hence had relatively few possible cryptographic keys. The version used by the German military added one sophistication: a *plugboard* (or *steckerbrett*) inserted between the input keyboard and the rotor scrambler, and also between the scrambler and the output light bulbs. The plugboard is also a simple substitution cipher, mapping L letters to another – different – letter, and the remaining $26 - 2L$ to themselves. However, this further cipher could easily be changed manually and was therefore part of the cryptographic key. For $L = 10$ plugs, the number of possibilities is $150, 738, 274, 937, 250$, and the total number of possible settings is thus $158, 962, 555, 217, 826, 360, 000$ and even 26^2 times more counting the reference positions.

Breaking the code entails finding, for a given ciphertext, the *rotor settings* (the choice of rotors, their order, the reference position of the two leftmost rotors, and their initial position) and the *plugboard configuration*. Those settings can be seen as the cryptographic key that one needs to recover in order to decipher the messages. However, if every component contributes to combinatorial number of keys, the rotor position is the most important. Indeed, the reference positions do not affect the first letters of the message. As long as the middle rotor does not advance, the text is unchanged. Moreover, the reference position of the first rotor can be deduced from when the text starts being gibberish. The plugboard configuration only affects part of the letters, for instance with six plugs, fourteen letters are not changed. Therefore given a correct guess on the order and position for the rotors, setting up the reference position arbitrary (say to AA) and the plugboard to the identity, would be sufficient

3 Only the left and middle rotors have a turnover notch.
4 Later versions introduced two more rotors to choose from, raising this number to ${}^3P_5 \times 26^5 = 721882560$.

to decrypt a small portion of the message. Therefore, verifying that this guess is correct is relatively easy, as well as deducing the reference position and the plugboard configuration. Finally, the rotor ordering (and choice thereof) was often guessed by other means than computation, or the attack was repeated for every possible order until it succeeded.

4 A Constraint Programming Model

In this section we introduce a constraint model that emulates the Enigma machine, and can be used to break its code. This model assumes that the rotors and their order are known. Since it is actually unknown, the problem might have to be solved $3! = 6$ (resp. ${}^3P_5 = 60$) times for three rotor wheels (resp. three out of five).

Let **ciphertext** be a string of K letters; **rotor** be the rotor wiring, whereby **rotor**$[j][x]$ is the output letter, on input x, of rotor j in its reference position; and **reflector** be the reflector wiring, whereby **reflector**$[x]$ is the output of the reflector on input x. In the following, we use N for the size of the alphabet and M for the number of rotor wheels (respectively 26 and 3 for Enigma). Moreover, we write $[a, b]$ for the discrete interval $\{a, a+1, \ldots, b\}$ and $[b]$ as a shortcut for $[0, b-1]$. The model uses four sets of variables:

$$\begin{aligned}
\textbf{plaintext} : plaintext_i &\in [N] & \forall i \in [K] \\
\textbf{key} : key_j &\in [N] & \forall j \in [M] \\
\textbf{ref} : ref_j &\in [N] & \forall j \in [M-1] \\
\textbf{plugboard} : plug[x] &\in [N] & \forall x \in [N]
\end{aligned}$$

The variables **plaintext** stand for the original message. The other variables stand for the settings of the machine used to cipher it: the variables **key** stand for the rotor position (rotor wheel j was advanced by key_j steps) the variables **ref** stand for the reference position of the two leftmost rotor wheels (rotor $j+1$ advances one step when rotor j returns to position ref_j), and the variables **plugboard** stand for the plugboard connection ($plug[x]$ is the letter "steckered" to x by the plugboard). In a nutshell it ensures that an Enigma machine with rotors **rotor** that have been setup with reference position **ref**, in initial position **key** and with plugboard configuration **plugboard** ciphers **plaintext** to **ciphertext**. We use the auxiliary variables $\mathbf{x}$ with $x_{i,j} \in [N]\ \forall i \in [K], \forall j \in [2M+2]$, with $x_{i,j}$ standing for the output of the scrambling device j for the letter at position i in the plaintext. The scrambling devices are ordered as explained in Section 3: plugboard first, then the three rotors, followed by the reflector, the three rotors backwards, and finally the plugboard again. We also use the auxiliary variables $\mathbf{p}$ with $p_{i,j} \in [K + K/N^j], \forall i \in [K], \forall j \in [M]$, to represent the total number of steps that the $j+1$-th rotor wheel has advanced when reading the $i+1$-th letter. Finally, the auxiliary variables **offset** with, for $j \in [2]$, $offset_j$ represent the reduction in number of steps to advance for rotor j to reach its reference position ref_j for the first time.[5]

In the reminder of the section, we will extensively use the *Element* constraint [4]:

▶ **Definition 6** (*Element*)**.** *Let* $\mathbf{x} = \{x_1, \ldots, x_K\}$ *be a set of variables, and* k, y *be two variables. The constraint* $Element(\mathbf{x}, k, y)$ *is the pair* (S, R) *where the relation* R *contains all assignments of* $S = \mathbf{x} \cup \{k, y\}$ *satisfying the predicate* $x_k = y$*.*

When we write the expression "x_k", with k a variable and $\mathbf{x} = x_1, \ldots, x_n$ an array of variables, this should be read as an extra variable y constrained with $Element(\mathbf{x}, k, y)$. Similarly, for any arithmetic operator $\oplus \in \{+, -, *, /, \mathrm{mod}\}$, the expression "$x_1 \oplus x_2$" should be read as an extra variable y with the constraint defined by the predicate $x_1 \oplus x_2 = y$.

[5] We include these variables for completeness, although they are often both set to the constant 0 (`A`).

4.1 Forward Rotor Model

If $\mathbf{rotor}[j][x]$ is the letter read after going through rotor j from input x, then after advancing k turns, the rotor produces the letter $(\mathbf{rotor}[j][(x+k) \bmod N] - k) \bmod N$ on the same input. The relation between the input letter $x_{i,j}$ and the output letter $x_{i,j+1}$ of the forward traversal of rotor j can therefore be encoded as follows:

$$p_{i,0} = key_0 + i \qquad \forall i \in [K] \qquad (1)$$

$$offset_j = \begin{cases} -ref_j & \text{if } ref_j \leq key_j \\ N - ref_j & \text{otherwise} \end{cases} \qquad \forall j \in [M-1] \qquad (2)$$

$$p_{i,j} = key_j + \frac{p_{i,j-1} + offset_{j-1}}{N} \qquad \forall i \in [K], \forall j \in [1, M] \qquad (3)$$

$$x_{i,j+1} = (\mathbf{rotor}[j][(x_{i,j} + p_{i,j}) \bmod N] - p_{i,j}) \bmod N \qquad \forall i \in [K], \forall j \in [M] \qquad (4)$$

Constraints (1, 2 and 3) channel the auxiliary variables $\mathbf{p}$, where $p_{i,j}$ stands for the number of times the rotor j has turned starting for position 0 when reading the i-th letter, to the initial positions **key** of the rotors and to their reference position **ref** via the variables **offset**. Constraints (4) represent the substitution cipher used with input letter $x_{i,j}$ and output letter $x_{i,j+1}$: if rotor j advanced p steps, the wire that initially connected the letter α to the letter β now connects the letter $\alpha - p$ to the letter $\beta - p$ (modulo $N = 26$).

4.2 Reflector Model & Backward Rotor Model

The reflector is also a substitution cipher, but static (it does not change from a letter to the next), symmetric ($\forall x, \forall y\ \mathbf{reflector}[x] = y \iff \mathbf{reflector}[y] = x$) and antireflexive ($\forall x\ \mathbf{reflector}[x] \neq x$). Constraints 5 model the relation between the input $x_{i,M+1}$ of the reflector (the signal corresponding to the i-th letter after going through the all rotors forward) and its output $x_{i,M+2}$ with another *Element* constraint. Then, the signal travels through the rotors, but backward. Constraints 6 are similar as for the forward pass.

$$x_{i,M+1} = \mathbf{reflector}[x_{i,M}] \qquad \forall i \in [K] \qquad (5)$$

$$x_{i,M+j+1} = (\mathbf{rotor}[j][(x_{i,M+j+2} + p_{i,M-j}) \bmod N] - p_{i,M-j}) \bmod N \qquad \forall i \in [K], \forall j \in [M] \qquad (6)$$

4.3 Plugboard Model

Finally, we need to model the plugboard of the military version. Since it is composed of L plugs, each one connecting two letters, it is a symmetric permutation with $N-2L$ *fixed points*, i.e., it leaves $N-2L$ letters unchanged. Unlike the reflector (which is fully known) or the rotors (for which the only unknowns are their positions), the plugboard is not known for the attacker: its configuration is part of the cryptographic key to be computed during the attack. It is encoded as a vector $\mathbf{plugboard} = \langle plug[1], \ldots, plug[N] \rangle$ of variables with domain $[N]$ where $plug[x]$ stands for the letter mapped to letter x, and the following constraints:

$$\textsc{AllDifferent}(\mathbf{plugboard}) \tag{7}$$

$$(plug[x] = y) \Leftrightarrow (plug[y] = x) \qquad \forall x, y \in [N] \tag{8}$$

$$\sum_{x=1}^{N}(plug[x] = x) = N - 2L \tag{9}$$

$$plug[plaintext_i] = x_{i,0} \qquad \forall i \in [K] \tag{10}$$

$$plug[\mathbf{ciphertext}[i]] = x_{i,2M+1} \qquad \forall i \in [K] \tag{11}$$

Constraints (8), (9) and (9) ensure that the plugboard is in a legal configuraton by stating that **plugboard** is a permutation (7); with $N - 2L$ identities (9)[6]; and is symmetric (8). Constraints (10) represent the transformation of the input by the plugboard, and Constraints (11) the final transformation, also by the plugboard, yielding the ciphertext as output.

4.4 Breaking the Code

So far, the model introduced in this section emulates the Enigma machine: a solution stands for a plaintext **x** whose cipher with message key **key** and plugboard **plugboard** is the given ciphertext. However, there are too many consistent assignments of **x**, **key** and **plugboard** to consider enumerating the solutions in order to find the original text.

In order to actually break the code, we use the same technique used by cryptanalyst at Bletchley Park in the 1940s. We suppose that we are somehow given a "crib", that is, a portion of deciphered message.[7] For instance, suppose that we know that the plaintext corresponding to the portion of ciphertext `IRSJYTCORS` starting at position s is in fact `CONSTRAINT`. Plaintext and ciphertext can therefore be aligned as shown in Table 1.

Table 1 A crib: an alignment of plaintext and ciphertext.

s	$s+1$	$s+2$	$s+3$	$s+4$	$s+5$	$s+6$	$s+7$	$s+8$	$s+9$
C	O	N	S	T	R	A	I	N	T
I	R	S	J	Y	T	C	O	R	S

Given a string of plaintext **T** and a string of ciphertext **C** such that $|\mathbf{T}| = |\mathbf{C}|$ starting at position s corresponding to a crib, we can find compatible initial rotor positions **key**, and plugboard configurations **plugboard**, by solving the model introduced in this section[8] with $K = |\mathbf{C}|$, Constraints (1–11), as well as the equalities:

$$plaintext_i = \mathbf{T}[i] \qquad \forall i \in [s, s + K] \tag{12}$$

This model may have several solutions, and only one corresponds to the actual cryptographic key. In practice, however, even small cribs can have a reasonable number of solutions, and verifying them manually can be done by deciphering the rest of the ciphertext with the same key. Experimental evaluations show that solving this model using standard CSP solvers is

[6] The number of connections L went from 6 to 10 depending on Enigma's versions. The equality "$plug[x] = x$" is read as its natural conversion from Boolean to $\{0, 1\}$.

[7] Cribs were obtained by guessing that a word or a sentence was likely to be in the message, and using the fact that a letter is never ciphered to itself to align that portion of plaintext with the ciphertext.

[8] Notice that the definition of several constraints must take into account the positional offset s.

not trivial (see Section 7). In the two following sections, we review how the code was broken in practice and show how those ideas can be mapped to the notion of consistency. These observations directly lead to an efficient CP based method to break Enigma's code.

5 Breaking Enigma's Code: The British Method

The method developped by Alan Turing (and improved by Gordon Welchman) computes the rotor position and the plugboard configuration used to cipher the message. It ignores the reference positions, however, and actually fails if there was a turnover (i.e., if the middle rotor advanced when ciphering the crib). A very advanced machine for its time, called "Bombe" was built for that purpose at Bletchley Park. The Bombe was not only capable of quickly simulating several rotor positions in parallel, but it could also either rule out a message key, or (partially) compute a plugboard configuration consistent with that message key.

The method used by British cryptanalists also relied on acquiring a crib, that is, a string $\mathbf{T}$ of original text aligned to a string $\mathbf{C}$ of same length in the ciphertext, as shown in Table 1. A *menu* for the Bombe was then extracted from the crib:

▶ **Definition 7** (Menu). *Given a string of plaintext* $\mathbf{T}$ *and a string of ciphertext* $\mathbf{C}$ *such that* $|\mathbf{T}| = |\mathbf{C}|$*, the* menu *obtained from matching them is a graph with one vertex per letter in* $\mathbf{T} \cup \mathbf{C}$*, and an edge for each pair of matched letters* $\mathbf{T}[i], \mathbf{C}[i]$ *labelled by their position, i.e.,* $G = (\mathbf{T} \cup \mathbf{C}, \{(\{\mathbf{T}[i], \mathbf{C}[i]\}, i) \mid i \in [|\mathbf{T}|]\})$*. We write* $N_G(x) = \{(y, i) \mid (\{x, y\}, i) \in G\}$ *for the neighborhood of letter* x *in the menu. Notice that it carries the edge labels.*

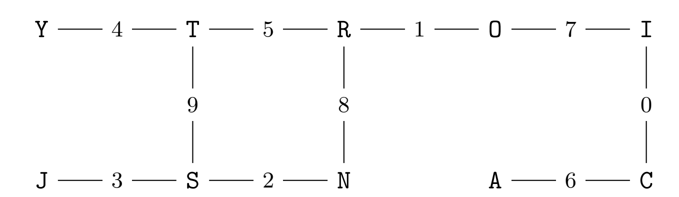


Figure 3 Illustration of the Bombe's menu from the crib in Table 1.

Figure 3 illustrates the menu extracted from the crib of Example 1.[9] An edge, say $(\{A, C\}, 6)$, of the menu indicates the letter `A` (resp. `C`) at position 6 is ciphered to `C` (resp. `A`). Now, observe that given some positions for the rotors, the Bombe can compute the scramble function (i.e., permutation of the alphabet) $S_i : [N] \mapsto [N]$ corresponding to the Enigma machine, ignoring the plugboard. It is then possible to make some inference on the configuration of the plugboard, as sketched in Figure 4. Indeed, suppose that the plugboard associates `A` to `B`. We can compute $S_i(B)$, and in particular $S_6(\texttt{B})$, let it be `R`. Now thanks to the crib, we know that the cipher for `A` at position 6 is `C`, therefore the plugboard must associate `R` to `C`. This process can be repeated starting from any edge of the menu ending in `R` or `C`, yielding more deductions. Eventually, a contradiction might be found, or a fixed point might be reached. This is Turing and Welchman's inference rule, which can be formalised as follows:

▶ **Definition 8** (Plugboard configuration). *Let* $\mathcal{P}$ *be a collection of subsets of* $[N]$ *such that* $|p| \leq 2 \ \forall p \in \mathcal{P}$*. We say that* $\mathcal{P}$ *is* valid *if and only if elements of* $\mathcal{P}$ *are pairwise disjoint and contains no more than* L *pairs and no more than* $N - 2L$ *singletons. We say that* $\mathcal{P}$ *is* complete *if and only if* $\bigcup_{p \in \mathcal{P}} p = [N]$*. The collection* $\mathcal{P}$ *is valid and complete if and only if it corresponds to a legal plugboard configuration where, for every* $\{\alpha, \beta\} \in \mathcal{P}$ *the letters* α *and* β *are connected, and for every* $\{\alpha\} \in \mathcal{P}$*, the letter* α *is not changed by the plugboard.*

[9] For the sake of the example, we let the first index s be 0

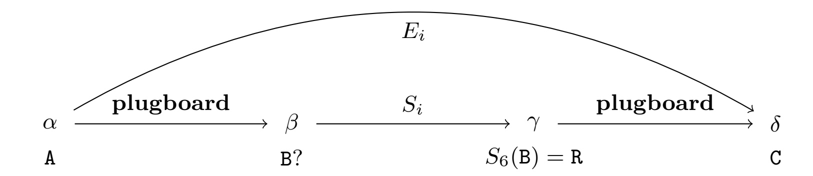


Figure 4 Illustration of Enigma's encryption scheme: the input character (here α) is mapped to β by the plugboard; then scrambled to γ by going through the rotors forward, the reflector and the rotors backward; and finally mapped to δ by the plugboard.

▶ **Proposition 9** (Inference rule). *Let S be a scrambler (Enigma without the plugboard) and G a menu. If $\mathcal{P}$ is the valid and complete plugboard configuration used during encryption of G, then, for every $\{\alpha, \beta\} \in \mathcal{P}$:*
- *for every $(\delta, i) \in N_G(\alpha)$, $\{\delta, S_i(\beta)\} \in \mathcal{P}$, and*
- *for every $(\delta, i) \in N_G(\beta)$, $\{\delta, S_i(\alpha)\} \in \mathcal{P}$.*

Proof. Suppose that the first rule does not hold, i.e., $\{\alpha, \beta\} \in \mathcal{P}$, $\exists(\delta, i) \in N_G(\alpha)$ such that $\{\delta, S_i(\beta)\} \notin \mathcal{P}$. The letter α is mapped to β by the plugboard and is scrambled to $S_i(\beta)$. However, since there is an edge $(\{\alpha, \delta\}, i)$ in the menu, we know that the encryption setup was such that the input letter α at position i yields letter δ. Therefore, the plugboard must match the letters $S_i(\beta)$ and δ. If there is no $p \in \mathcal{P}$ such that $\beta \in p$, then $\mathcal{P}$ is not complete, and otherwise, it is not consistent, hence a contradiction.

The second rule follows from the same reason but starting from letter β. ◀

This inference rule can be used to discard a guessed plugboard connection. The Bombe is an electro-mechanical device that automatises this process. It is composed of several clones of the Enigma machine, each capable of emulating the encryption of all 26 letters in parallel for a given rotor setting, and wires for every possible plugboard connection. Given a message key **key**, and a menu, the Bombe was initialised by setting up one of the clones to emulate the scrambler S_i for every edge label i in the menu. Then a connection $\{\alpha, \beta\}$ could be "guessed" by sending a current flow through the corresponding wire into the Bombe. The inference rule described in Proposition 9 would then be applied as the current flowed through the input β of the clone S_i, for each $(\delta, i) \in N_G(\alpha)$. The current would in turn flow to the connection between $S_i(\beta)$ and δ and again to some clones of Enigma as indicated in the menu. This can either result in a fixed point where $\mathcal{P}$ is closed under that rule, or yields an invalid plugboard configuration. In the latter case, the connection $\{\alpha, \beta\}$ can be ruled out, and another guess can be made. If there exists a letter for which no matching is possible, the message **key** is ruled out and the same process is repeated for another message key. Otherwise, **key** is a candidate key and a (partial) plugboard configuration is given by the connection in which the current flows.

▶ **Definition 10** (The Bombe method). *The Bombe made it possible, starting from a tentative connection $\{\alpha, \beta\}$, to enforce the inference rule described in Proposition 9 until either it* fails *($\mathcal{P}$ is no longer valid), or* succeeds *(it does not fail and reaches a fixed point). In the former case, the connection $\{\alpha, \beta\}$ can be ruled out.*

The Bombe method *denotes the process where, for each rotor position, every possible plugboard connection involving a letter in the crib is ruled out if the process above fails. The Bombe method causes a* stop *if and only if it reaches a fixed point where every letter can appear in at least one connection. In that case, the current rotor position might be the correct one (the message key) and it is checked by other means. Otherwise, this key is ruled out and the process resume with another of the 26^3 positions.*

▶ **Lemma 11.** *Let $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ be the CSP of Section 4. If the rule of Proposition 9 deduces the connection $\{\delta, \gamma\}$ from menu G and $\mathcal{P} = \{\{\alpha, \beta\}\}$ then enforcing AC on $(\mathcal{X}, \mathcal{D}|_{plug[\alpha] \leftarrow \beta}, \mathcal{C})$, reduces the domain of $plug[\delta]$ to $\{\gamma\}$ and of $plug[\gamma]$ to $\{\delta\}$.*

Proof. Suppose that the rule of Proposition 9 deduces the connection $\{\delta, \gamma\}$. We assume first that the first rule triggered and hence $(\delta, i) \in N_G(\alpha)$ and $S_i(\beta) = \gamma$.

Constraints (1 – 6) and (10 – 12) are all functional: when all but one of their variables are fixed, the last variable has a single consistent value.

Since $(\delta, i) \in N_G(\alpha)$, the letter α in the plaintext of the crib is matched to δ in the ciphertext at position i. By enforcing *AC* on Constraint 12, we have $D(plaintext_i) = \{\alpha\}$, and since $D(plug[\alpha]) = \{\beta\}$, by enforcing *AC* on Constraint (10) we have $D(x_{i,0}) = \{\beta\}$.

Moreover, since **key** is constant, so is **p**, by enforcing *AC* on Constraint (1) since only one non-constant variable remains. As a result, enforcing *AC* on Constraints (4) reduces the domains of variables $x_{i,1}, x_{i,2}$ and $x_{i,3}$ to single values. The same is true for Constraint (5) and Constraints (6). The variable $x_{i,2M+1}$ is therefore assigned, and it must be to value $S_i(\beta) = \gamma$. Enforcing *AC* on Constraint (11) sets the domain of variable $plug[\delta]$ to $\{\gamma\}$. Finally, Constraint (8) set the domain of $plug[\gamma]$ to $\{\delta\}$.

If it was the second rule that triggered, the same demonstration applies, although the chain of propagations to consider goes in the reverse direction: from $x_{j,2M+1}$ to $x_{j,0}$. ◀

The implication in Lemma 11 is not an equivalence simply because in some cases, Constraints 7, 8 and 9 might trigger and more connections could then be deduced via propagation. For instance if $L = 1$ and $\alpha \neq \beta$, then all variables in **plugboard** would be assigned by propagation of Constraint 9. The following theorem is also an implication for the same reasons, and in this case these constraints are even more likely to be relevant since the domains are reduced while enforcing *SAC*.

▶ **Theorem 12.** *If the Bombe method does not produce a stop, then enforcing singleton arc consistency on the model described in Section 4, with the variables* **key** *assigned, also fails.*

Proof. Let $\mathcal{CN} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ be the CSP described in Section 4. In particular, **plugboard** $\subseteq \mathcal{X}$, however, **plaintext** and **key** are constant under the theorem's hypothesis.

Suppose that the Bombe method rules out a connection $\{\alpha, \beta\}$. It means that the rule of Proposition 9 produces an invalid plugboard configuration $\mathcal{P}$. However, from Lemma 11 we know that for every $\{\alpha, \beta\} \in \mathcal{P}$, after enforcing *AC* on the CSP $(\mathcal{X}, \mathcal{D}|_{plug[\alpha] \leftarrow \beta}, \mathcal{C})$, all but the value β (res.p α) are removed from $D(plug[\alpha])$ (resp. $D(plug[\beta])$). It entails that there exist two elements of $\mathcal{P}$ that are not disjoint, e.g., $\{\alpha, \beta\} \in \mathcal{P}$ and $\{\alpha, \gamma\} \in \mathcal{P}$. In this case, the corresponding domain $D(plug[\alpha])$ is wiped out.

Therefore, every connection that is ruled out by the Bombe method is also ruled out by enforcing *SAC*. The Bombe method fails if all the connections involving a particular letter have been ruled out. If this happens for letter α, then the domain of the variable $plug[\alpha]$ is emptied, and therefore *SAC* fails. ◀

The Bombe can therefore be seen as a CSP solver that branches on the variables **key**, and enforces singleton arc consistency on leaves of this search tree. When *SAC* fails, the rotor position corresponding to that branch is ruled out, otherwise the machine *stops*. Then, the (partial) plugboard configuration is verified manually. If it cannot be extended to a configuration that correctly deciphers the message, then this rotor position is ruled out as well, and the Bombe resumes its search with a new assignment of **key**. It was therefore important to reduce the likelihood of such *false stops*.

6 Breaking Enigma's Code: The Polish Method

The Enigma code was actually first broken, before the war, by Polish mathematicians Marian Rejewski, Jerzy Różycki and Henryck Zygalski [5], who also succesfully reconstructed, from partial and indirect intelligence, the Enigma machine and built mechanical devices to emulate it. Their insights later proved essential to the British effort. Turing and Welchman's method relied on a crib and would guess the rotor position and the plugboard configuration used to encrypt the message regardless of how it was chosen (in particular we shall see that this rotor position is different to the one given in the daily settings). The method of the Polish Cipher Bureau, on the other hand, relied on intercepting several messages sent with the same daily settings, and would compute not directly the message key, but the daily settings.

Using repeatedly the same encryption key would be a serious security flaw. Operators were thus instructed to randomly choose a 3-letter *message key*. This key (e.g., `SAT`) would be encrypted *twice*[10] using the rotor position indicated in the daily settings (e.g., the text `SATSAT` would be ciphered using the key `CSP` to some 6-letter ciphertext). Then, the machine would be reset to position `SAT` and the message woud be ciphered using that message key, and both ciphers (6-letter prefix and text) sent in the same message. The receiver would then set its own Enigma machine to postion `CSP`, decipher the prefix, reset its machine to the obtained position `SAT`, and finally decrypt the message. This method still had the weakness of using the same key (here, `CSP`) to encrypt all messages keys for a given day. Rejewski devised a first method, involving a dedicated machine, the *cyclometer*, that partially automated the design of a lookup table (the *cards catalog*) from which the daily rotor positions could easily be found, provided several prefixes encrypted with the same daily settings [1].

The procedure was therefore upgraded: the sender chose a *plaintext key* besides the message key. Then, the rotors were positioned according to the daily rotor settings *shifted by the plaintext key*. In other words, the actual encryption key was equal to *rotor settings* + *plaintext key*, where "+" stands for the modular, component-wise addition. This key was used to encrypt the message key and it was sent in plaintext with the encrypted message. For instance, if the rotor setting is `CSP`, the chosen plaintext key `MIP`, then the rotors would be put in position `OAE` = `CSP` + `MIP` to cipher the chosen message key (say `SAT`). The text `SATSAT` would yield, for instance, `DGFAGX`. Then the rotors were reset to position `SAT` and the actual message was ciphered to "**ciphertext**". Finally, the message sent would be:

`MIP` `DGFAGX` **ciphertext**

The receiver would add the plaintext key to the daily setting to recover the actual rotor position, then, as previously, decipher the message key, reset the machine to the corresponding rotor position and decipher the message. The difference with the previous practice, however, was that the rotor position used to cipher the 6-letter prefix was never twice the same. The fact that the message key was repeated twice, however, proved nonetheless to be a fatal flaw.

When rotors and plugboard are in a given configuration, Enigma corresponds to a symmetric permutation, although the permutation changes with every letter since the rotors advance. Let $E_i^{\texttt{XYZ}}$ stand for the permutation applied by Enigma to the i-th letter with initial rotor position `XYZ`.[11] Consider the prefix `MIP DGFAGX` and let $\mathbf{key} = \{key_1, key_2, key_3\}$ denote the unknown rotor setting common to all messages of a given day. We know that a

[10] This practice was abandoned May 1st 1940, hence making the Polish attack irrelevant.
[11] The plugboard is ignored here.

letter (say x) is mapped to the letter D in $E_1^{\textbf{key+MIP}}$ and to A in $E_4^{\textbf{key+MIP}}$. Moreover, since $E_1^{\textbf{key+MIP}}$ is symmeric, it maps A to x. Therefore, the composition $E_1^{\textbf{key+MIP}}E_4^{\textbf{key+MIP}}$ maps D to A and vice versa. Because the plugboard is unknown we do not learn anything from that.

However, observe that $E_2^{\textbf{key+MIP}}E_5^{\textbf{key+MIP}}$ transforms the letter G to itself. Mathematician Henryk Zygalski noticed that this was relevant because the plugboard changes the letter but conserve the fixed points of permutation $E_2^{\textbf{key+MIP}}E_5^{\textbf{key+MIP}}$ (an element unchanged by the permutation), and because not all rotor settings have such fixed points [5].

A device, "the Bomba",[12] was designed and built by the Polish Cipher Bureau to go over all of the 26^3 rotor settings, and record which settings could allow such a fixed point in one of the three composed permutations. Only 40% of the rotor positions had a composition fixed point. A set of 26 perforated sheets (known as the *Zygalski sheets*) were to be produced for each of the 6 possible rotor orders.[13] Each sheet corresponds to a letter, standing for the first letter of the unknown rotor position, and contains a 26×26 Boolean matrix standing for whether the second and third letters could extend the first letter to a position allowing a fixed point. We denote $Z[\alpha, \beta, \gamma]$ the fact that there is a hole at positions β, γ in the Zygalski sheet for letter α, which is true if and only if the rotor position $\alpha\beta\gamma$ has a composition fixed point (and hence may encode the two occurrences of a letter in the prefix to the same code).

Consider now a prefix MIP VNEVSX. There is a fixed point at position $(1, 4)$, hence the Zygalski sheets allowed some inference: one would first guess the order of the rotors and a first letter key_1 indicating the position of the first rotor. Then, let $\alpha = key_1 + \mathtt{M}$. The α-th sheet would be taken from the box standing for the chosen order. The matrix on that sheet (shifted by I and P in the respective dimensions[14]) thus stands for the possible values of key_2 and key_3. This narrows down the number of possibilities by 60%. Now, suppose that a message with prefix: SMT DGFAGX is intercepted the same day. This prefix also has a fixed point, but at position $(2, 5)$. Therefore, the same reasoning applies, using sheet $key_1 + \mathtt{S} + 1$ in the same box, but shifted by M and T. The "+1" models that the fixed point is on the 2nd and 5th letters, which is equivalent to observing it on the 1st and 4th letters, however with the position of the left rotor advanced one step. The subset of holes that allow both fixed points is obtained by shining light through the two sheets, properly shifted and aligned. Usually, a dozen messages including a fixed point (and as many sheets) were necessary to narrow down the number of possibilities to either 0, in which case the guess was proven wrong; or 1, in which case the rotor position **key** and the rotor order could be easily retrieved.

This method does not take the plugboard into consideration. However, since 14 letters were not affected by the plugboard[15], it is possible to reconstruct the message manually. It also fails in case a turnover of the middle rotor happens before the end of the 6-letter prefix.

Aligning several Zygalski sheets corresponds to solving the following CSP:

▶ **Definition 13** (Zygalski's CSP). *We call Zygalski's CSP the constraint satisfaction problem with set of variables* $\textbf{key} = \{key_1, key_2, key_3\}$*, and for each message with plaintext key* $\alpha\beta\gamma$ *containing a fixed point at positions* $(1 + i, 4 + i)$*, a constraint given by the predicate* $Z[key_1 + \alpha + i, key_2 + \beta, key_3 + \gamma]$.

▶ **Theorem 14.** *The letter* α *for the position of the first rotor is refuted by the Zygalski sheets method if and only if no pair of instantiations* $\{key_2 \leftarrow \beta, key_3 \leftarrow \gamma\}$ *has* $key_1 \leftarrow \alpha$ *as path support in Zygalski's CSP.*

[12] An aggregate of six Enigma machines, one for each permutations of the prefix.
[13] The process was long and difficult, even with the Bomba: only 2 sets had been completed when the invasion of Poland began, but the sheets were eventually finalized at Blechtley Park.
[14] The sheets had 25 repeated rows and columns to allow for shifting modulo 26.
[15] Only 6 plugs were used when the Polish began to attack Enigma.

Proof. By definition, a hole with coordinates β, γ will let the light shine through all sheets, if and only if $\{key_1 \leftarrow \alpha, key_2 \leftarrow \beta, key_3 \leftarrow \gamma\}$ is consistent with every constraint of Zygalski's CSP, that is, if and only if $key_1 \leftarrow \alpha$ is a path support of $\{key_2 \leftarrow \beta, key_3 \leftarrow \gamma\}$. ◀

The process of superimposing several Zygalski sheets corresponding to a guess of the letter α for the position of the first rotor is an efficient method to compute the 2-tuples which have $key_1 \leftarrow \alpha$ as path consistent support. Moreover, since there are only three variables in the network, if $key_1 \leftarrow \alpha$ is a path support of $\{key_2 \leftarrow \beta, key_3 \leftarrow \gamma\}$ then $key_2 \leftarrow \beta$ is a path support of $\{key_1 \leftarrow \alpha, key_3 \leftarrow \gamma\}$ and $key_3 \leftarrow \gamma$ is a path support of $\{key_1 \leftarrow \alpha, key_2 \leftarrow \beta\}$. Therefore path consistency can be achieved by only checking the values of a single variable in this way, there is a solution if and only there is a tuple with a path support.

Path consistency is usually only applied to binary constraints. However, the definition we use naturally extends this consistency to non-binary constraints and in that context, the analogy stands. For instance, consider two Zygalski sheets: one allowing the keys `ABC`, `ADE`, `BBE` and `BDC` and one allowing the keys `ABE`, `ADC`, `BBC` and `BDE`. There is no solution, and indeed the Zygalski CSP is not *PC* (e.g., the consistent and valid assignemnt $\{key_1 \leftarrow \mathtt{A}, key_2 \leftarrow \mathtt{B}\}$ cannot be extend to the third variable), yet the CSP is *AC* and *SAC*.

7 Experimental Evaluation

We ran experiments to verify the impact of *SAC* on this problem. The constraint model was implemented using the toolkit Choco [8].[16] To emulate the Bombe, we force the heuristic to select the variables standing for the positions of the rotors (**key**) before other variables. In the version denoted `Choco+SAC`, we run *SAC* only once these variables are all assigned. When *SAC* does not fail, which corresponds to a stop of the Bombe, or in the default version denoted `Choco`, we let the constraint solver either find a solution for the variables **plugboard**, or prove that no solution exists for the current rotor position. For both methods we treated every variable in **ref** as the constant 0 as was done in the methods we discussed previously. Not doing so would increase the search space, and the number of solutions, by a factor 26^2.

In order to generate benchmark instances, we used many cribs of length 12 from wikipedia texts that we ciphered using a random position of the rotors I, II and III of the first Enigma machine that was introduced in 1930 [9], and a random reflector.

Fast:	`E K M F L G D Q V Z N T O W Y H X U S P A I B R C J`
Middle:	`A J D K S I R U X B L H W T M C Q G Z N P Y F V O E`
Slow:	`B D F H J L C P R T X V Z N Y E I W G A K M U S Q O`
Reflector:	`P R Y Z L X O S Q K J E N M G A I B H W V U T F C D`

We selected 260 cribs, so that we uniformly cover a range of values for the following parameters: *size of the menu* (size), i.e., the number of vertices in the graph of the extracted menu and *number of cycles* (#cycle). The idea is that sparse or acyclic menus produce more stops. In particular Turing made an analysis of how many stops would occur according to the values of these parameters [2]. Therefore, higher number of cycles and lower menu size should make the problem easier. Moreover, we ignored menus with four or more connected components. We average the results on instances with same size and number of cycles.

[16] The source code is available here: `https://gitlab.laas.fr/vantuori/sacnigma.git`.

Table 2 Results of `Choco` and `Choco+SAC` on a range of cribs.

Crib					`Choco`			`Choco+SAC`		
#inst.	#cycle	size	#stops	#sol.	#solved	CPU	#fail	#solved	CPU	#fail
10	0	12	128.4	17912	3	378	388790.7	10	1197	17448.6
10	0	13	242.8	12214	7	901	1036519.1	10	922	17334.3
10	0	14	5.6	593	7	989	968841.6	10	866	17570.4
10	0	15	2.0	26	8	438	414718.2	10	607	17574.0
10	0	16	1.0	14	8	1208	1251729.9	10	813	17575.0
10	1	11	68.6	124111	5	475	542790.6	10	908	17507.5
10	1	12	156.2	33912	6	958	1435607.2	10	751	17421.3
10	1	13	1.8	1182	5	604	693048.2	10	1061	17574.2
10	1	14	1.9	166	8	606	580758.1	10	748	17574.1
10	1	15	1.0	77	8	369	350508.9	10	560	17575.0
10	1	16	1.0	40	8	1057	1239245.6	10	832	17575.0
10	2	10	18.2	165837	4	604	585009.0	9	1048	17557.8
10	2	11	63.0	45071	7	637	958325.9	10	796	17515.3
10	2	12	1.2	2141	7	476	544253.6	10	734	17574.8
10	2	13	2.0	4002	4	391	378688.0	10	793	17574.0
10	2	14	1.0	85	9	391	376638.7	10	578	17575.0
10	2	15	1.0	39	7	715	798622.9	10	755	17575.0
10	2	16	1.0	32	9	352	349036.6	10	618	17575.0
10	3	9	7.6	240639	6	437	338330.8	9	907	17573.2
10	3	10	27.7	192315	2	451	659440.0	10	1267	17551.4
10	3	11	1.1	10001	5	719	987604.6	10	959	17574.9
10	3	12	2.1	19165	4	560	679398.8	10	916	17573.9
10	3	13	1.0	341	7	341	331790.3	10	696	17575.0
10	3	14	1.0	127	9	390	359362.0	10	534	17575.0
10	3	15	1.0	486	8	358	342439.2	10	528	17575.0
10	3	16	1.0	31	9	524	513744.6	10	612	17575.0

All experiments were run on 4 cluster nodes, with Intel Xeon CPU E5-2695 v4 2.10GHz cores running Linux Ubuntu 16.04.4. We ran both models for one hour or until completion on every instance. We report in Table 2 the characteristics of the cribs, the average number of solutions (#sol.), and the average number of solutions with distinct rotor positions, that is, the number of "stops" (#stops). Then, for both methods, we report the number of instances solved (#solved), that is, where all solutions have been enumerated within the one hour cutoff, the average CPU time (in seconds) over solved instances (CPU) and the average number of fails during search (#fail). We can observe that using *SAC* significantly improves the model: `Choco` solves about 69% of the instances in less than one hour, whereas `Choco+SAC` solves 99% of the instances in about 15 minutes in average. The number of fails of `Choco+SAC` shows clearly that, as expected, constraint propagation does not actually cut the search tree for the rotor positions. All of the 17576 possible keys are explored. However, it also shows that in the few cases where *SAC* does not fail (the stops), virtually every singleton arc consistent permutation of the plugboard is consistent with the crib. Indeed the solver does not fail, and the number of solutions may become relatively large in that case. On the other hand, the number of fails of `Choco` shows a more conventional picture where plugoard configurations are ruled out by a blend of propagation and search.

The number of stops is lower than we would expect of the Bombe by about one or two orders of magnitude for low number of cycles. This is because we count only rotor positions for which a consistent plugboard configuration exists, whereas the Bombe stops as soon as a

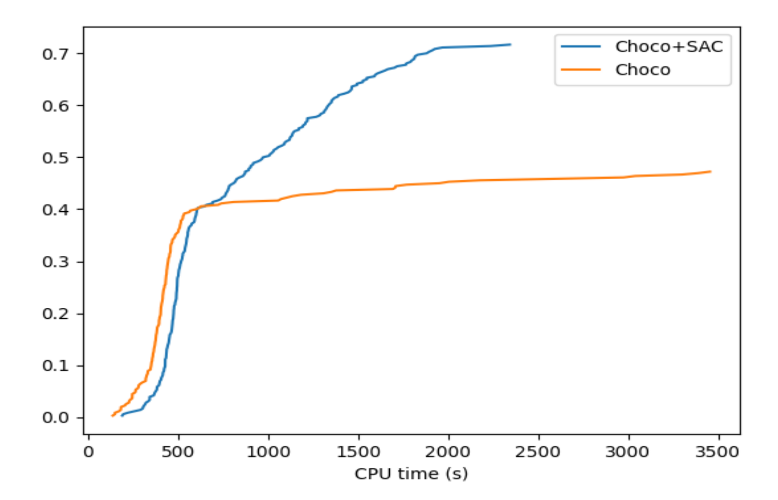


Figure 5 Cumulative probability to break the code in less than X seconds.

(slightly weaker form of) *SAC* can be enforced. Moreover, enforcing *SAC* on the constraint model takes advantage of propagation of the constraints modeling the plugboard (e.g., the AllDifferent constraint). Finally, the Bombe only takes into account the letters of the menu, i.e., it does not check that these extra letters too must have a legal plug connection.

Interestingly, we observe that the number of solutions tends to be larger for denser and more cyclic menus, even though the number of consistent rotor positions decreases, as we would expect from the Bombe. It shows that there are many more consistent plugboard configurations in this case. Overall, those parameters have a lower impact on the constraint model as they seem to have had on the Bombe. Overall, the method `Choco+SAC` is not extremely fast, but it is very robust. The graph in Figure 5 shows that in the most favorable cases, not enforcing singleton arc consistency can be the most efficient approach.

8 Conclusion

We have shown that the method designed by Alan Turing and Gordon Welchman at Bletchley Park to break the Enigma code has uncanny similarities with applying Singleton Arc Consistency on a constraint satisfaction problem modeling the machine. Experiments show that indeed, Singleton Arc Consistency significantly reduces the computation time required to decipher a message. Moreover, the method designed by Marian Rejewski et al. before that can also be related to achieving Path Consistency on another constraint satisfaction problem.

References

1 Chris Christensen. Polish Mathematicians Finding Patterns in Enigma Messages. *Mathematics Magazine*, 80(4):247–273, 2007. URL: `http://www.jstor.org/stable/27643040`.

2 Cipher A. Deavours and Louis Kruh. The Turing Bombe: Was it Enough? *Cryptologia*, 14(4):331–349, 1990.

3 Romuald Debruyne and Christian Bessiere. Some Practicable Filtering Techniques for the Constraint Satisfaction Problem. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI)*, 1997.

4 Pascal Van Hentenryck and Jean-Philippe Carillon. Generality versus Specificity: An Experience with AI and OR Techniques. In *Proceedings of the 7th National Conference on Artificial Intelligence (AAAI)*, 1988.

5 Wladyslaw Kozaczuk. *Enigma. How the German Machine Cipher Was Broken, and how It Was Read by the Allies in World War II.* University Publications of America, 1984.

6 Alan K. Mackworth. Consistency in Networks of Relations. In Bonnie Lynn Webber and Nils J. Nilsson, editors, *Readings in Artificial Intelligence*, pages 69–78. Morgan Kaufmann, 1981. doi:10.1016/B978-0-934613-03-3.50009-X.

7 Ugo Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Information Sciences*, 7:95–132, 1974. doi:10.1016/0020-0255(74)90008-5.

8 Charles Prud'homme, Jean-Guillaume Fages, and Xavier Lorca. *Choco Solver Documentation*. TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S., 2016. URL: http://www.choco-solver.org.

9 Wikipedia contributors. Enigma rotor details, 2021. URL: https://en.wikipedia.org/wiki/Enigma_rotor_details.